Juggling Functions Inside a Database

Mahmoud Abo Khamis
LogicBlox Inc.
mahmoud.abokhamis@logicblox.com

Hung Q. Ngo
LogicBlox Inc.
hung.ngo@logicblox.com

Atri Rudra
University at Buffalo, SUNY
atri@buffalo.edu

ABSTRACT

We define and study the **F**unctional **A**ggregate **Q**uery (FAQ) problem, which captures common computational tasks across a very wide range of domains including relational databases, logic, matrix and tensor computation, probabilistic graphical models, constraint satisfaction, and signal processing. Simply put, an FAQ is a declarative way of defining a new function from a database of input functions.

We present InsideOut, a dynamic programming algorithm, to evaluate an FAQ. The algorithm rewrites the input query into a set of easier-to-compute FAQ sub-queries. Each subquery is then evaluated using a worst-case optimal relational join algorithm. The topic of designing algorithms to optimally evaluate the classic multiway join problem has seen exciting developments in the past few years. Our framework tightly connects these new ideas in database theory with a vast number of application areas in a coherent manner, showing potentially that – with the right abstraction, blurring the distinction between data and computation – a good database engine can be a general purpose constraint solver, relational data store, graphical model inference engine, and matrix/tensor computation processor all at once.

The InsideOut algorithm is very simple, as shall be described in this paper. Yet, in spite of solving an extremely general problem, its runtime either is as good as or improves upon the best known algorithm for the applications that FAQ specializes to. These corollaries include computational tasks in graphical model inference, matrix/tensor operations, relational joins, and logic. Better yet, InsideOut can be used within any database engine, because it is basically a principled way of rewriting queries. Indeed, it is already part of the LogicBlox database engine, helping efficiently answer traditional database queries, graphical model inference queries, and train a large class of machine learning models inside the database itself.

This work was partly supported by NSF grant CCF-1319402 and by DARPA under agreement #FA8750-15-2-0009. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright thereon.

1. INTRODUCTION

The following fundamental problems from diverse domains share a common algebraic structure involving (generalized) sums of products.

Example 1. (Matrix Chain Multiplication (MCM)) Given a series of matrices $\mathbf{A}_1, \dots, \mathbf{A}_n$ over some field $\mathbb{F}$, where the dimension of $\mathbf{A}_i$ is $p_i \times p_{i+1}$, $i \in [n]$, we wish to compute the product $\mathbf{A} = \mathbf{A}_1 \cdots \mathbf{A}_n$. The problem can be reformulated as follows. There are $n+1$ variables $X_1, \dots, X_{n+1}$ with domains $\mathsf{Dom}(X_i) = [p_i]$, for $i \in [n+1]$. For $i \in [n]$, matrix $\mathbf{A}_i$ can be viewed as a function of two variables

$$\psi_{i,i+1} : \mathsf{Dom}(X_i) \times \mathsf{Dom}(X_{i+1}) \to \mathbb{F},$$

where $\psi_{i,i+1}(x, y) = (\mathbf{A}_i)_{xy}$. The MCM problem is to compute the output function

$$\varphi(x_1, x_{n+1}) = \sum_{x_2 \in \mathsf{Dom}(X_2)} \cdots \sum_{x_n \in \mathsf{Dom}(X_n)} \prod_{i=1}^{n} \psi_{i,i+1}(x_i, x_{i+1}).$$

Example 2. (Maximum A Posteriori (MAP) queries in probabilistic graphical models (PGMs)) Consider a discrete graphical model represented by a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$. There are n discrete random variables $\mathcal{V} = \{X_1, \dots, X_n\}$ on finite domains $\mathsf{Dom}(X_i)$, $i \in [n]$, and $m = |\mathcal{E}|$ *factors*

$$\psi_S : \prod_{i \in S} \mathsf{Dom}(X_i) \to \mathbb{R}_+, \ S \in \mathcal{E}.$$

A typical inference task is to compute the marginal MAP estimates, written in the form

$$\varphi(x_1, \dots, x_f) = \max_{x_{f+1} \in \mathsf{Dom}(X_{f+1})} \cdots \max_{x_n \in \mathsf{Dom}(X_n)} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S).$$

Example 3. (Conjunctive query in RDBMS) Consider a schema with the following input relations: $R(a, b)$, $S(b, c)$, $T(c, a)$, where for simplicity let us say all attributes are integers. Consider the following query:

```
SELECT R.a
FROM R, S, T
WHERE R.b = S.b AND S.c = T.c AND T.a = R.a;
```

The above query can be reformulated as follows. Relation $R(a, b)$ is modeled by a function $\psi_R(a, b) \to \{\mathsf{true}, \mathsf{false}\}$, where $\psi_R(a,b) = \mathsf{true}$ iff $(a,b) \in R$, and relations $S(b,c)$ and $T(c,a)$ are modeled by similar functions $\psi_S(b,c), \psi_T(c,a)$. Now, computing the above query basically corresponds to computing the function $\varphi(a) \to \{\mathsf{true}, \mathsf{false}\}$, defined as:

$$\varphi(a) = \bigvee_b \bigvee_c \psi_R(a,b) \wedge \psi_S(b,c) \wedge \psi_T(c,a).$$

Example 4. (# Quantified Conjunctive Query (#QCQ)) Let Φ be a first-order formula of the form

$$\Phi(X_1,\dots,X_f) = Q_{f+1}X_{f+1}\cdots Q_n X_n \left(\bigwedge_{R \in \mathsf{atoms}(\Phi)} R \right),$$

where $Q_i \in \{\exists, \forall\}$, for $i > f$. The #QCQ problem is to *count* the number of tuples in relation Φ on the free variables $X_1,\dots,X_f$. To reformulate #QCQ, construct a hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ as follows: $\mathcal{V}$ is the set of all variables $X_1,\dots,X_n$, and for each $R \in \mathsf{atoms}(\Phi)$ there is a hyperedge $S = \mathsf{vars}(R)$ consisting of all variables in R. The atom R can be viewed as a function indicating whether an assignment $\mathbf{x}_S$ to its variables is satisfied by the atom; namely $\psi_S(\mathbf{x}_S) = 1$ if $\mathbf{x}_S \in R$ and 0 otherwise.

For each $i \in \{f+1,\dots,n\}$ we define an aggregate operator

$$\bigoplus^{(i)} = \begin{cases} \max & \text{if } Q_i = \exists, \\ \times & \text{if } Q_i = \forall. \end{cases}$$

Then, the #QCQ problem above is to compute the *constant* function

$$\varphi = \sum_{x_1 \in \mathsf{Dom}(X_1)} \cdots \sum_{x_f \in \mathsf{Dom}(X_f)} \bigoplus^{(f+1)}_{x_{f+1} \in \{0,1\}} \cdots \bigoplus^{(n)}_{x_n \in \{0,1\}} \prod_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S).$$

It turns out that these and dozens of other fundamental problems from constraint satisfaction (CSP), databases, matrix operations, PGM inference, logic, coding theory, and complexity theory can be viewed as special instances of a generic problem we call the **F**unctional **A**ggregate **Q**uery, or the FAQ problem, which we define next. (See [2, 6] for many more examples.)

Throughout the paper, we use the following convention. Uppercase X_i denotes a variable, and lowercase x_i denotes a value in the domain $\mathsf{Dom}(X_i)$ of the variable. Furthermore, for any subset $S \subseteq [n]$, define

$$\mathbf{X}_S = (X_i)_{i \in S}, \qquad \mathbf{x}_S = (x_i)_{i \in S} \in \prod_{i \in S} \mathsf{Dom}(X_i).$$

In particular, $\mathbf{X}_S$ is a tuple of variables and $\mathbf{x}_S$ is a tuple of specific values with support S. The input to FAQ is a set of functions and the output is a function computed using a series of aggregates over the variables and input functions. More specifically, for each $i \in [n]$, let X_i be a variable on some discrete domain $\mathsf{Dom}(X_i)$, where $|\mathsf{Dom}(X_i)| \geq 2$. The FAQ problem is to compute the following function

$$\varphi(\mathbf{x}_{[f]}) = \bigoplus^{(f+1)}_{x_{f+1} \in \mathsf{Dom}(X_{f+1})} \cdots \bigoplus^{(n)}_{x_n \in \mathsf{Dom}(X_n)} \bigotimes_{S \in \mathcal{E}} \psi_S(\mathbf{x}_S), \quad (1)$$

where

- $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is a multi-hypergraph. $\mathcal{V} = [n]$ is the index set of the variables X_i, $i \in [n]$. Overloading notation, $\mathcal{V}$ is also referred to as the set of variables.
- The set $F = [f]$ is the set of *free variables* for some integer $0 \leq f \leq n$. Variables in $\mathcal{V} - F$ are called *bound variables*. (Free and bound are logic terminologies. Free variables are group-by variables in database nomenclature.)
- $\mathbf{D}$ is a fixed domain, such as $\{0,1\}$, $\mathbb{R}^+$, $\mathbb{Z}$.
- For every hyperedge $S \in \mathcal{E}$, $\psi_S : \prod_{i \in S} \mathsf{Dom}(X_i) \to \mathbf{D}$ is an *input* function (also called a *factor*).
- For every bound variable $i > f$, $\oplus^{(i)}$ is a binary (aggregate) operator on the domain $\mathbf{D}$.
- And, for each bound variable $i > f$ either $\oplus^{(i)} = \otimes$ or $(\mathbf{D}, \oplus^{(i)}, \otimes)$ forms a commutative semiring (with the same $\mathbf{0}$ and $\mathbf{1}$). Informally, this means that we can do addition and multiplication over $\mathbf{D}$ and still remain in the same set.

  If $\oplus^{(i)} = \otimes$, then $\oplus^{(i)}$ is called a *product aggregate*; otherwise, it is a *semiring aggregate*. (We assume that there is at least one semiring aggregate.)

Because for $i > f$ every variable X_i has its own aggregate $\bigoplus^{(i)}$ over all values $x_i \in \mathsf{Dom}(X_i)$, in the rest of the paper we will write $\bigoplus^{(i)}_{x_i}$ to mean $\bigoplus^{(i)}_{x_i \in \mathsf{Dom}(X_i)}$.

We will refer to φ as an FAQ-*query*. We use FAQ-SS to denote the special case when there is a *S*ingle *S*emiring aggregate, i.e. $\oplus^{(i)} = \oplus, \forall i > f$, and $(\mathbf{D}, \oplus, \otimes)$ is a semiring [6].

Example 5. (Aggregate query in RDBMS) Consider the following query over relations $R(a,b)$, $S(a,c)$, $T(b,c,d,e)$, $U(d,f)$, $V(e,f)$, $W(e,g)$, $Y(f,h)$, where all attributes are integers:

```
SELECT R.b, U.d, sum(W.e)
FROM R, S, T, U, V, W, Y
WHERE R.a = S.a AND R.b = T.b AND S.c = T.c
      AND T.d = U.d AND T.e = V.e AND W.e = V.e
      AND U.f = V.f AND Y.f = V.f GROUP BY R.b, U.d;
```

We now explain how the above query can be reduced to an FAQ instance. Relation $R(a,b)$ is modeled with a function $\psi_R(a,b) \to \{0,1\}$, where $\psi_R(a,b) = 1$ iff $(a,b) \in R$. Similarly, we can think of relations S, T, U, V, and Y as functions ψ_S, ψ_T, ψ_U, ψ_V, ψ_Y, with $\{0,1\}$ values. We single out one relation $W(e,g)$ where the modeling is different: $\psi_W(e,g) = e$ if $(e,g) \in W$ and 0 otherwise. The corresponding FAQ-query is

$$\varphi(b,d) = \sum_a \sum_c \sum_e \sum_f \sum_g \sum_h \psi_R \psi_S \psi_T \psi_U \psi_V \psi_W \psi_Y$$

(For readability, we did not write the argument lists of the functions ψ_R, ψ_S, etc. They should be obvious from context.) Note that a tuple in the output of the aggregate query has the schema $(b, d, \varphi(b,d))$. The corresponding hypergraph is shown in Fig. 1a. The set of free variables is $F = \{b,d\}$. The domain is $\mathbf{D} = \mathbb{Z}$, the set of integers. Note

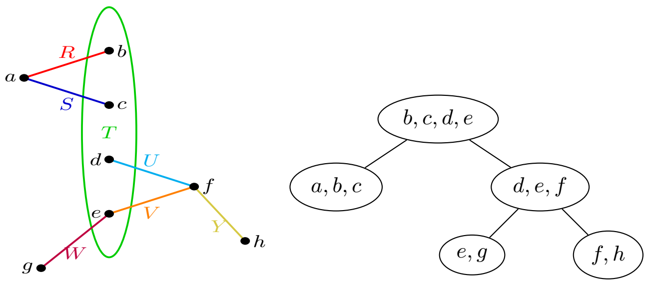
A triple $(\mathbf{D}, \oplus, \otimes)$ is a *commutative semiring* if $\oplus$ and $\otimes$ are commutative binary operators over $\mathbf{D}$ satisfying the following: (1) $(\mathbf{D}, \oplus)$ is a commutative monoid with an additive identity, denoted by $\mathbf{0}$. (2) $(\mathbf{D}, \otimes)$ is a commutative monoid with a multiplicative identity, denoted by $\mathbf{1}$. (3) $\otimes$ distributes over $\oplus$. (4) For any element $e \in \mathbf{D}$, $e \otimes \mathbf{0} = \mathbf{0} \otimes e = \mathbf{0}$.

also that the above reduction to FAQ still works if we replace sum by another aggregate, e.g., max.

In order to explain later the connection of InsideOut to query rewriting, we also write the above query in LogiQL, an extension of Datalog supported by the LogicBlox engine [7]:

```
Q[b, d] = s <- agg<<s = total(e)>> R(a,b), S(a,c),
    T(b,c,d,e), U(d,f), V(e,f), W(e,g), Y(f,h).
```

In the above, `agg` is short for aggregate, `total` is equivalent to `sum` in SQL, the notation `Q[b,d]=s` means that the head predicate is `Q(b,d,s)` where `(b,d)` is a key, hence the query computes `Q(b,d,sum(e))`.



(a) Query hypergraph (b) Tree decomposition

Figure 1: Query from Example 5

The above example illustrates several important points. First, when we defined the FAQ problem we did not specify how the input and output factors are represented. The representation choice turns out to make a huge difference in computational complexity [2]. However, in practical applications the representation is usually the obvious one: an input factor $\psi_S(\mathbf{X}_S)$ can be thought of as a table of tuples $[\mathbf{x}_S, \psi_S(\mathbf{x}_S)]$, with the implicit assumption that if $\mathbf{x}_S$ is not in the table then its ψ_S-value is $\mathbf{0}$. (This is the additive identity $\mathbf{0}$ of the domain $\mathbf{D}$.) Second, the reduction to FAQ is only at the syntax level. No real data conversion is necessary. All the data we need to obtain the functions ψ_R, ψ_T etc. are already in the input relations. Third, in the mathematical definition of $\varphi(b, d)$ above, the domains of all variables are integers and so we have *infinite* sums. We could have restricted all variables to their active domains; but that is not necessary because summing over all integers or over the active domains give identical answer: tuples not present are assumed to have values $\mathbf{0}$.

Now that we have established the scope of FAQ, in the remainder of this paper we show a perhaps surprising result that an FAQ problem can be solved by *one* simple yet efficient algorithm. The algorithm can be implemented as a set of ordinary database queries. The runtime matches or improves upon the best known runtimes in many application areas that the FAQ framework captures. The runtime depends on the order of variable aggregates in the FAQ expression, which naturally leads us to the question of how to re-order those aggregates to obtain the best runtime without changing the semantic meaning of the expression.

2. THE INSIDEOUT ALGORITHM

Parts of this section will be familiar to readers who have been exposed to elementary graphical models [24]. There are, however, a couple of ideas that are taken from new developments in database theory [29, 28, 3] that are likely not known in the graphical model literature. For each factor ψ_S, define its *size* to be the number of non-zero points under its domain: $|\psi_S| := \left|\{\mathbf{x}_S \mid \psi_S(\mathbf{x}_S) \neq \mathbf{0}\}\right|$.

Basic variable elimination. To describe the intuition, we first explain InsideOut as it applies to the special case of FAQ-SS (or SumProd). The idea behind variable elimination [17, 37, 36] is to 'fold' common factors, exploiting the distributive law:

$$\bigoplus_{x_{f+1}} \cdots \bigoplus_{x_n} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$$

$$= \bigoplus_{x_{f+1}} \cdots \bigoplus_{x_{n-1}} \bigotimes_{S\in\mathcal{E}-\partial(n)} \psi_S(\mathbf{x}_S) \otimes \underbrace{\left(\bigoplus_{x_n} \bigotimes_{S\in\partial(n)} \psi_S(\mathbf{x}_S) \right)}_{\text{new factor } \psi_{U_n-\{n\}}},$$

where the equality follows from the fact that $\otimes$ distributes over $\oplus$, $\partial(n)$ denotes all edges incident to n in $\mathcal{H}$ and $U_n = \cup_{S\in\partial(n)} S$. Assume for now that we can somehow efficiently compute the intermediate factor $\psi_{U_n-\{n\}}$. Then, the resulting problem is another instance of FAQ-SS on a modified multi-hypergraph $\mathcal{H}'$, constructed from $\mathcal{H}$ by removing vertex n along with all edges in $\partial(n)$, and *adding back* a new hyperedge $U_n - \{n\}$. Recursively, we continue this process until all variables $X_n, \ldots, X_{f+1}$ are eliminated. Textbook treewidth-based results for PGM inference are obtained this way [24]. In the database context (i.e. given an FAQ-query over the Boolean semiring), the intermediate result $\psi_{U_n-\{n\}}$ is essentially an intermediate relation of a query plan, the folding technique exploiting distributive law corresponds to "pushing the aggregate down" the query plan [13].

Introducing the indicator projections. While correct, basic variable elimination as described above is potentially not very efficient for sparse input factors, i.e. factors where the number of non-zero entries is much smaller than the product of the active domain sizes. This is because the product that was factored out of the scope of X_n might annihilate many entries of the intermediate result $\psi_{U_n-\{n\}}$, while we have spent so much time computing $\psi_{U_n-\{n\}}$. For example, for an $S \notin \partial(n)$ such that $S \subseteq U_n$ and tuple $\mathbf{y}_S$ such that $\psi_S(\mathbf{y}_S) = \mathbf{0}$, we do not need to compute the entries $\psi_{U_n-\{n\}}(\mathbf{x}_{U_n-\{n\}})$ for which $\mathbf{y}_S = \mathbf{x}_S$: those entries will be eliminated later anyhow. The idea is then to only compute those $\psi_{U_n-\{n\}}(\mathbf{x}_{U_n-\{n\}})$ values that will "survive" the other factors later on. One simple way to achieve this would be to compute, for each $S \in \mathcal{E} - \partial(n)$, an "indicator factor" that checks if $\psi_S(\mathbf{x}_S)$ is $\mathbf{0}$ or not. Formally, for any two sets $T \subseteq S$, and a given factor ψ_S, the function $\psi_{S/T} : \prod_{i\in T} \mathsf{Dom}(X_i) \to \mathbf{D}$ defined by

$$\psi_{S/T}(\mathbf{x}_T) := \begin{cases} \mathbf{1} & \exists \mathbf{x}_{S-T} \text{ s.t. } \psi_S(\mathbf{x}_T, \mathbf{x}_{S-T}) \neq \mathbf{0} \\ \mathbf{0} & \text{otherwise} \end{cases}$$

is called the *indicator projection* of ψ_S onto T. Using indicator factors, InsideOut computes the following factor when marginalizing X_n away:

$$\psi_{U_n-\{n\}}(\mathbf{x}_{U_n-\{n\}}) = \bigoplus_{x_n} \left[\left(\bigotimes_{S\in\partial(n)} \psi_S \right) \otimes \left(\bigotimes_{\substack{S\notin\partial(n),\\ S\cap U_n \neq \emptyset}} \psi_{S/S\cap U_n} \right) \right]. \quad (2)$$

Another minor tweak is the observation that, if there is a hyperedge $S \in \mathcal{E} - \partial(n)$ for which $S \subset U_n$, then we do not use the indicator projection $\psi_{S/S\cap U_n}$: we can use ψ_S itself to compute the intermediate factor $\psi_{U_n-\{n\}}$, and then remove ψ_S from $\mathcal{H}'$.

Example 6. We explain how the ideas above are implemented in Example 5. First, the order in which we choose to eliminate variables might have a huge effect on the runtime. For now, let us assume that we somehow decided to rewrite $\varphi(b,d)$ using the following variable order, where we trace the first couple of steps of the InsideOut algorithm *without* the indicator projection: (Example 10 later explains how this order is related to the tree decomposition in Fig. 1b.)

$$\begin{aligned}
\varphi(b,d) &= \sum_c\sum_a\sum_e\sum_f\sum_g\sum_h \psi_R\psi_S\psi_T\psi_U\psi_V\psi_W\psi_Y \\
&= \sum_c\sum_a\sum_e\sum_f\sum_g \psi_R\psi_S\psi_T\psi_U\psi_V\psi_W \underbrace{\sum_h \psi_Y(f,h)}_{\psi_1(f)} \\
&= \sum_c\sum_a\sum_e\sum_f\sum_g \psi_R\psi_S\psi_T\psi_U\psi_V\psi_W\psi_1 \\
&= \sum_c\sum_a\sum_e\sum_f \psi_R\psi_S\psi_T\psi_U\psi_V\psi_1 \underbrace{\sum_g \psi_W(e,g)}_{\psi_2(e)} \\
&= \sum_c\sum_a\sum_e\sum_f \psi_R\psi_S\psi_T\psi_U\psi_V\psi_1\psi_2
\end{aligned}$$

The first two steps are straightforward, where we eliminated g and h. In LogiQL, these intermediate factors are computed with the following two rules

```
psi1[f] = s1 <- agg<<s1 = count()>> Y(f,h).
psi2[e] = s2 <- agg<<s2 = total(e)>> W(e,g).
```

The mathematical abstraction corresponds to rewriting a query into a series of smaller queries. Next, we explain how the indicator projection works when we eliminate variable f. Out of the remaining factors ψ_R,ψ_S,ψ_T,ψ_U,ψ_V,ψ_1,and ψ_2,the following factors contain f: $\psi_U(d,f)$ $\psi_V(e,f)$ and $\psi_1(f)$. If we were to multiply them together and marginalize away f, we would create a new factor $\psi_3(e,d) = \sum_f \psi_U\psi_V\psi_1$ over variables $\{e,d\}$. However, two other factors have variables that overlap with $\{e,d\}$, namely $\psi_T(b,c,d,e)$ and $\psi_2(e)$. For ψ_T, we include its indicator projection $\psi_{T/\{e,d\}}$ in computing ψ_3. (We will see later in Example 7 how including $\psi_{T/\{e,d\}}$ can actually speed up the computation of ψ_3 asymptotically.) For ψ_2, we can include ψ_2 itself. (Recall the minor tweak we mentioned above.) Overall, we end up with the following definition of ψ_3:

$$\psi_3(e,d) = \sum_f \psi_U \cdot \psi_V \cdot \psi_1 \cdot \psi_2 \cdot \psi_{T/\{e,d\}}.$$

In LogiQL, this sub-result is computed with two rules:

```
proj1(d,e) <- T(b,c,d,e). // projection rule
psi3[e,d] = s3 <- agg<<s3 = total(s1*s2)>> U(d,f),
    V(e,f), psi1[f] = s1, psi2[e] = s2, proj1(d,e).
```

After eliminating f, we are left with the following

$$\begin{aligned}
\varphi(b,d) &= \sum_c\sum_a\sum_e \psi_R\psi_S\psi_T\psi_3 \\
&= \sum_c\sum_a \psi_R\psi_S \sum_e \psi_3\psi_T \\
&= \sum_c\sum_a \psi_R\psi_S \underbrace{\sum_e \psi_3\psi_T\psi_{R/\{b\}}\psi_{S/\{c\}}}_{\psi_4(b,c,d)}
\end{aligned}$$

leading to the following LogiQL rules

```
proj2(b) <- R(a,b).
proj3(c) <- S(a,c).
psi4[b,c,d] = s4 <- agg<<s4 = total(s3)>>
   psi3[e,d] = s3, T(b,c,d,e), proj2(b), proj3(c).
```

At this point, we have 3 factors left $\psi_R(a,b)$, $\psi_S(a,c)$, and $\psi_4(b,c,d)$. We eliminate a then c straightforwardly:

$$\sum_c\sum_a \psi_R\psi_S\psi_4 = \sum_c \psi_4 \underbrace{\sum_a \psi_R\psi_S\psi_{4/\{b,c\}}}_{\psi 5(b,c)} = \sum_c \psi_4\psi_5.$$

Note that $\psi_{4/\{b,c\}}$ has values in $\{0,1\}$ although ψ_4 can have any value in $\mathbb{Z}$. The final LogiQL rules are

```
proj4(b,c) <- psi4[b,c,d] = s4. // indicator projection
psi5[b,c] = s5 <- agg<<s5 = count()>>
                  R(a,b), S(a,c), proj4(b,c).
output[b,d] = t <- agg<<t = total(s4*s5)>>
                  psi4[b,c,d] = s4, psi5[b,c] = s5.
```

The general FAQ problem. The above strategy does not care if the variable aggregates where the same or different: As long as $(\mathbf{D}, \oplus^{(n)}, \otimes)$ is a semiring, we can fold the common factors and eliminate X_n. Thus, InsideOut works almost as is for a general FAQ instance (as opposed to FAQ-SS). Finally, when $\oplus^{(n)} = \otimes$ we simply swap the two (identical) operators:

$$\begin{aligned}
\varphi(\mathbf{x}_{[f]}) &= \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigoplus_{x_n}^{(n)} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S) \\
&= \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigotimes_{x_n\in\mathsf{Dom}(X_n)} \bigotimes_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S) \\
&= \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigotimes_{S\in\mathcal{E}} \bigotimes_{x_n\in\mathsf{Dom}(X_n)} \psi_S(\mathbf{x}_S) \\
&= \cdots \bigoplus_{x_{n-1}}^{(n-1)} \bigotimes_{S\notin\partial(n)} \underbrace{(\psi_S(\mathbf{x}_S))^{|\mathsf{Dom}(X_n)|}}_{\psi'_S} \bigotimes_{S\in\partial(n)} \underbrace{\bigotimes_{x_n} \psi_S(\mathbf{x}_S)}_{\psi_{S-\{n\}}}.
\end{aligned}$$

We are left with an FAQ-instance whose hypergraph is exactly $\mathcal{H}' = \mathcal{H} - \{n\}$: the hypergraph obtained form $\mathcal{H}$ by removing vertex n from the vertex set and all incident hyperedges. The sub-problems are of the form of *product marginalizations* of individual factors ψ_S for $S \in \partial(n)$, each of which can be computed in linear time in $|\psi_S|$. The product marginalization step is algorithmically much easier because it does not create the intermediate factor $\psi_{U_n-\{n\}}$. As for $S \notin \partial(n)$, we replace ψ_S by the power factor $\psi'_S(\mathbf{x}_S) = (\psi_S(\mathbf{x}_S))^{|\mathsf{Dom}(X_n)|}$, which can be done in linear time with a $\log|\mathsf{Dom}(X_n)|$ blowup using the repeated squaring algorithm. Note the key fact that this power is with respect to the product aggregate $\otimes$. In most (if not all) applications of FAQ, there is one additional property: most of the time, $\otimes$ is an idempotent operator over the active domain. For example, in the #QCQ problem $\otimes$ is the usual product operator and the domain that it aggregates over is $\{0,1\}$ (before there is a sum outside). In this case, $\psi'_S(\mathbf{x}_S) =$

$(\psi_S(\mathbf{x}_S))^{|\mathsf{Dom}(X_n)|} = \psi_S(\mathbf{x}_S)$, and we do not need to spend the linear nor log-blowup time. For more details on product idempotence, see [2].

FAQ sub-problems as natural joins. In the above we have explained how InsideOut breaks a big problem into smaller problems. In the product marginalization case, the sub-problems are easy to solve: they can be solved in linear time. The most difficult problems, however, are of the form (2). This is exactly an FAQ-query where we marginalize out only one variable, with the remaining variables free. Zooming in, problem (2) is of the form

$$\psi_{U_n - \{n\}}(\mathbf{x}_{U_n - \{n\}}) := \bigoplus_{x_n} \bigotimes_{F \in \mathcal{E}_n} \psi_F,$$

where $\mathcal{H}_n = (U_n, \mathcal{E}_n)$ is the sub-FAQ-query hypergraph. The problem is solved by computing $\psi_{U_n}(\mathbf{x}_{U_n}) := \bigotimes_{F \in \mathcal{E}_n} \psi_F$ first. Once the ψ_{U_n} is computed, marginalizing away X_n to obtain $\psi_{U_n - \{n\}}$ is trivial.

Computing the inner product is a natural join problem in disguise. Each input factor ψ_S is represented using a table of tuples of the form $[\mathbf{x}_S, \psi_S(\mathbf{x}_S)]$. Essentially, $\mathbf{x}_S$ is the (compound) key and $\psi_S(\mathbf{x}_S)$ is the value in this relation. Again, recall that entries not in the table have ψ_S-value $\mathbf{0}$. Hence, to compute ψ_{U_n} we can first join the tables ψ_S using only the key space. For each tuple $\mathbf{x}_{U_n}$ in the result of this join, we record the value $\psi_{U_n}(\mathbf{x}) = \prod_{F \in \mathcal{E}_n} \psi_F(\mathbf{x}_F)$. The runtime is dominated by the natural join's runtime.

Worst-case optimal join algorithms. Computing the natural join is a very well-studied problem with exciting new developments in the past decade or so. There are new *worst-case optimal* algorithms [35, 28, 29, 1] that operate quite differently from traditional query plans, in the sense that they no longer compute one pairwise join at a time, but instead process the query globally. While the vast majority of database engines today still rely on traditional query plans, new, complex data analytics engines are switching to worst-case optimal algorithms: LogicBlox's engine [7] is built on a worst-case optimal algorithm called LeapFrog Triejoin [35] (LFTJ), and the *Myria* data analytics platform supports a variant of LFTJ [12].

We briefly outline these results here. The generic form of the natural join problem can be posed in our hypergraph language as $Q = \bowtie_{F \in \mathcal{E}} R_F$, where $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ is the query hypergraph. The vertices of this hypergraph consist of all attributes. Each hyperedge $F \in \mathcal{E}$ corresponds to an input relation R_F whose attributes are F. The natural join problem can be thought of as a constraint satisfaction problem: each input relation R_F imposes a constraint where a tuple $\mathbf{x}_F$ satisfies the constraint if $\mathbf{x}_F \in R_F$. A tuple $\mathbf{x}$ on all variables $\mathcal{V}$ is an output of the join if the projection $\mathbf{x}_F$ satisfies R_F for all $F \in \mathcal{E}$.

LFTJ [35] can be viewed as *backtracking-search* algorithm, which was known some 50 years ago in the AI and constraint programming world [16, 20]. (In contrast, by saving intermediate results $\psi_{U_n - \{n\}}$ instead of re-computing them each time, InsideOut can be thought of as *dynamic programming*. The duality between backtracking search and dynamic programming is well-known [33].) LFTJ fixes some variable ordering $X_1, \ldots, X_n$ of the query Q, then performs "leap-frogging" to find the first binding x_1 that does not yet violate any constraints R_F; once x_1 is found, it looks for the first binding x_2 such that the partial tuple (x_1, x_2) does not violate any constraint. The algorithm proceeds this way until either a full binding $\mathbf{x}$ is constructed in which case $\mathbf{x}$ is an output, or no good binding is found. For example, if no feasible binding for x_3 is found, then the algorithm backtracks to the next good binding of x_2.

The first advantage of backtracking search is that it requires only $O(1)$-extra space: it does not cache any computation. The second advantage, amazingly, is that a join algorithm based on back-tracking search such as LFTJ or others in [28, 29] are worst-case optimal, in the sense that the algorithm runs in time bounded by the worst-case output size. To state the output size bound, we need the following notion. Define the *fractional edge cover polytope* $\mathsf{P}(\mathcal{H})$ associated with a hypergraph $\mathcal{H}$ to be the set of all vectors $\boldsymbol{\lambda} = (\lambda_F)_{F \in \mathcal{E}}$ satisfying the following linear constraints:

$$\boldsymbol{\lambda} \geq \mathbf{0}, \text{ and } \sum_{F \in \mathcal{E}, v \in F} \lambda_F \geq 1, \ \forall v \in \mathcal{V}.$$

A vector $\boldsymbol{\lambda} \in \mathsf{P}(\mathcal{H})$ is called a *fractional edge cover* of $\mathcal{H}$. The join output size is bounded above by $\prod_{F \in \mathcal{E}} |R_F|^{\lambda_F}$, for *any* $\boldsymbol{\lambda} \in \mathsf{P}(\mathcal{H})$. The best bound $\mathsf{AGM}(\mathcal{H})$, known as the AGM-bound [8, 21], is obtained by solving the linear program

$$\min\Big\{\sum_{F \in \mathcal{E}} \lambda_F \log_2 |R_F| : \boldsymbol{\lambda} \in \mathsf{P}(\mathcal{H})\Big\}. \tag{3}$$

Example 7. Consider the query computing ψ_3 in Example 6. The join query on the keys has the following shape: $Q = U(d, f) \bowtie V(e, f) \bowtie I(f) \bowtie J(e) \bowtie K(d, e)$. Then, $\mathsf{AGM}(Q) = |U|^{\lambda_{d,f}} |V|^{\lambda_{e,f}} |I|^{\lambda_f} |J|^{\lambda_e} |K|^{\lambda_{d,e}}$, where $\boldsymbol{\lambda}$ is a fractional edge cover of the query's hypergraph. Suppose all input relations have the same size N, then the optimal bound is obtained by setting $\lambda_{d,f} = \lambda_{d,e} = \lambda_{e,f} = 1/2$, and $\lambda_d = \lambda_e = 0$. Worst-case optimal algorithms run in time $\tilde{O}(N^{3/2})$ for this instance. Any traditional join-tree based plan runs in $\Omega(N^2)$-time for some input [28]. Moreover, without the indicator projection of $T(b, c, d, e)$, there would be no $K(d, e)$ above, the best edge cover would be $\lambda_{d,f} = \lambda_{e,f} = 1$, and the runtime would become $\Omega(N^2)$.

Runtime analysis. Let N denote the input size, $|\mathsf{output}|$ the output size, and K the set of $k \in [n]$ for which $\oplus^{(k)} \neq \otimes$ (note that $[f] \subseteq K$). Also, let $\mathsf{AGM}(Q_k)$ denote the AGM-bound on the kth sub-query's hypergraph $\mathcal{H}_k$. Then, it is not hard to show [2] that the runtime of InsideOut is

$$\tilde{O}\Big(N + \sum_{k \in K} \mathsf{AGM}(\mathcal{H}_k) + |\mathsf{output}|\Big). \tag{4}$$

The first term is input-preprocessing time, second is the total subproblem solving time, and third is the unavoidable output reporting time. From (4), we can write down a precise expression for the runtime of InsideOut. Minimizing the resulting (somewhat complicated) expression leads to the dynamic programming algorithm for the MCM problem and the FFT algorithm for the DFT (see [2] for details).

In the above discussion, we assumed that variables were eliminated in order $X_n, X_{n-1}, \ldots, X_1$. However, there is no reason to force InsideOut to follow this particular order. In particular, there might be a different variable ordering for which expression (4) is *a lot* smaller and the algorithm still works correctly on that ordering (see [2]). This is where the main technical contributions of our work in [2] begin. We need to answer the following two fundamental questions:

Question 1. How do we know which variable orderings are equivalent to the original FAQ-query expression?

Question 2. How do we find the "best" variable ordering among all equivalent variable orderings?

In the next two sections, we sketch how we answered the above two questions and followup questions in theory and in practice.

3. THEORETICAL CONTRIBUTIONS

To answer the above questions, we start with some definitions. A variable ordering σ is *φ-equivalent* iff permuting the variable aggregates of φ using σ gives an expression φ' that is *semantically-equivalent* to φ, i.e. that always returns the same output as φ *no matter* what the input is. Let $\mathsf{EVO}(\varphi)$ denote the set of all φ-equivalent variable orderings.

Example 8. The FAQ query φ' below is φ-equivalent.

$$\begin{aligned}\varphi &= \sum_a \sum_d \max_b \sum_c \psi_1(a,b)\psi_2(a,c)\psi_3(c,d),\\ \varphi' &= \sum_a \sum_c \sum_d \max_b \psi_1(a,b)\psi_2(a,c)\psi_3(c,d).\end{aligned}$$

This is because φ can be written as

$$\varphi = \sum_a \left[\left(\sum_d \sum_c \psi_2(a,c)\psi_3(c,d) \right) \left(\max_b \psi_1(a,b) \right) \right].$$

Now, for any $\sigma \in \mathsf{EVO}(\varphi)$, let $\mathcal{H}_k^\sigma$ denote the kth subquery's hypergraph when we run $\mathsf{InsideOut}$ on σ. Ideally, we would like to find σ minimizing the expression $\sum_{k\in K} \mathsf{AGM}(\mathcal{H}_k^\sigma)$. However, this expression is data-dependent and thus it is a bit difficult to handle in a mathematically clean way. We simplify our objective by approximating the bound (4) a little: we upperbound $\mathsf{AGM}(\mathcal{H}_k^\sigma)$ by the *fractional edge cover number* of the subgraph $\mathcal{H}_k^\sigma$, i.e. $\mathsf{AGM}(\mathcal{H}_k^\sigma) \le N^{\rho^*(\mathcal{H}_k^\sigma)}$, where $\rho^*(\mathcal{H}_k^\sigma) := \min\{\sum_{F\in\mathcal{E}} \lambda_F : \boldsymbol{\lambda} \in \mathsf{P}(\mathcal{H}_k^\sigma)\}$. Then, (4) is upperbounded by $\tilde{O}\left(N^{\max_{k\in K}\rho^*(\mathcal{H}_k^\sigma)} + |\mathsf{output}|\right)$; $\mathsf{InsideOut}$ on variable ordering σ runs in $\tilde{O}(N^{\mathsf{faqw}(\sigma)} + |\mathsf{output}|)$-time, where $\mathsf{faqw}(\sigma) := \max_{k\in K} \rho^*(\mathcal{H}_k^\sigma)$. Thus, to have the best runtime we would like to select an equivalent ordering σ with the smallest exponent $\mathsf{faqw}(\sigma)$, called the FAQ-*width* of an FAQ-query:

$$\mathsf{faqw}(\varphi) := \min\{\mathsf{faqw}(\sigma) \mid \sigma \in \mathsf{EVO}(\varphi)\} \tag{5}$$

Example 9. In Example 8, the original order in φ has an FAQ-width of 2, because eliminating c first corresponds to joining ψ_2 and ψ_3 in time $\Omega(N^2)$. In contrast, the order in φ' has an faqw of 1, allowing to evaluate φ in time $O(N)$.

To solve the optimization problem (5), the first problem we have to address is to precisely characterize the set $\mathsf{EVO}(\varphi)$. Our approach, sketched in Fig. 2, is to construct an *expression tree* of the FAQ query φ. The expression tree defines a partially ordered set on the variables called the *precedence poset*. Then, to complete the characterization of EVO, we show that every ordering in EVO is *component-wise equivalent* (CWE) to a linear extension of $\mathsf{LinEx}(P)$. (See [2] for details.) Thus, if we do not care about query complexity, we can take the orange path in Fig 2 and bruteforcedly compute an optimal variable ordering σ^*, run it through $\mathsf{InsideOut}$, for a total runtime of $\tilde{O}(N^{\mathsf{faqw}(\varphi)} + |\mathsf{output}|)$.

However, in some FAQ-framework's applications such as in graphical models, we cannot simply sweep query complexity under the rug. Moreover, computing the faqw is **NP**-hard because faqw is a strict generalization of the fractional hypertree width (fhtw), which is **NP**-hard [19] to compute. Hence, we find a good approximation for the faqw. This is the green path in Fig. 2: from the expression tree, we construct a tree decomposition for $\mathcal{H}$; then, from the GYO-elimination order of this tree decomposition we obtain a variable ordering $\bar{\sigma}$ for which we can show that $\mathsf{faqw}(\bar{\sigma}) \le \mathsf{faqw}(\sigma^*) + g(\mathsf{faqw}(\sigma^*))$, where g is any known approximation of fhtw (the best of which is due to Marx [26]).

Example 10. The variable ordering used earlier in Example 6 is a GYO-elimination order for the tree decomposition in Fig. 1b. (In GYO, when we eliminate variable X_n, the set U_n becomes a bag of the tree decomposition whose children are the bags corresponding to $\partial(n)$.) In particular, bags $\{f,h\}$, $\{e,g\}$, and $\{d,e,f\}$ resulted from eliminating h, g, and f respectively. The tree decomposition has width $\mathsf{fhtw} = 3/2$, same as the faqw of the variable ordering.

From these ideas, we obtained many corollaries, some of which are summarized in Table 1. The results in Table 1 span three areas: (1) CSPs and Logic; (2) PGMs and (3) Matrix operations. Except for joins, problems in area (1) need the full generality of FAQ formulation, where $\mathsf{InsideOut}$ either improves upon existing results or yields new results. Problems in area (2) can already be reduced to $\mathsf{FAQ\text{-}SS}$. Here, $\mathsf{InsideOut}$ improves upon known results since it takes advantage of Grohe and Marx's more recent fractional hypertree width bounds. Finally, problems in area (3) of Table 1 are classic. $\mathsf{InsideOut}$ does not yield anything new here, but it is intriguing to be able to explain the textbook dynamic programming algorithm for $\mathsf{Matrix\text{-}Chain\ Multiplication}$ [15] as an algorithm to find a good variable ordering for the corresponding FAQ-instance. The DFT result is a re-writing of Aji and McEliece's observation [6].

4. PRACTICAL IMPLICATIONS

In this section we address two questions the readers might have regarding $\mathsf{InsideOut}$: (1) hurdles one might face in a practical implementation of $\mathsf{InsideOut}$, and (2) whether practical implications are as good as what the theory says.

Additional hurdles and how to solve them. There are a couple of problems we have to solve to implement $\mathsf{InsideOut}$ effectively.

The first problem is, in real-world queries, we do not just have materialized predicates as inputs, we also have predicates such as $a < b$, $a + b = c$, negations and so on. These predicates do not have a "size." To solve this problem, one solution is to set the "size" of those predicates to be ∞ while computing the AGM-bound. For instance, if we have a subquery of the form $Q \leftarrow R(a,b), S(b,c), a+b=c$, where R and S are input materialized predicates of size N, then by setting the size of $a+b=c$ to be infinite, $\mathsf{AGM}(Q) = N^2$. This solution does not work for two reasons. (1) If we knew $a+b=c$, then it is easy to infer that $|Q| \le N$ and also to compute Q in time $\tilde{O}(N)$: scan over tuples in R, use $a+b=c$ to compute c, and see if $(b,c) \in S$. In other words, the AGM-bound is no longer tight. (2) This solution may give an ∞-bound when the output size is clearly bounded. Consider, for example, the query $Q \leftarrow R(a), S(b), a+b=c$; in this case, $\{a,b,c\}$ is the only hyperedge covering vertex c in the fractional edge cover. Our implementation at $\mathsf{LogicBlox}$

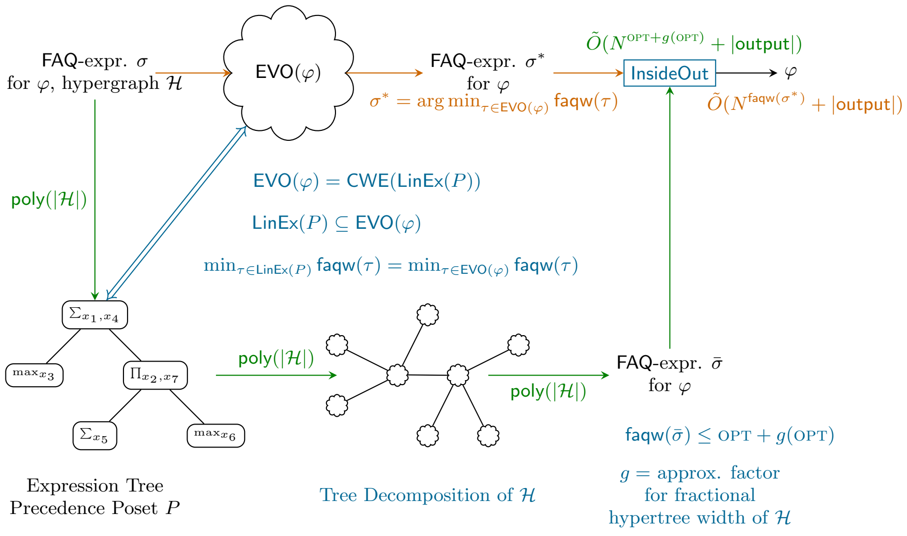


Figure 2: Sketch of main technical contributions

Problem	FAQ formulation	Previous Algo.	**Our Algo.**
#QCQ	$\sum_{(x_1,\dots,x_f)} \bigoplus^{(f+1)}_{x_{f+1}} \cdots \bigoplus^{(n)}_{x_n} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	No non-trivial algo	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
QCQ	$\bigoplus^{(f+1)}_{x_{f+1}} \cdots \bigoplus^{(n)}_{x_n} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$ where $\bigoplus^{(i)} \in \{\max, \times\}$	$\tilde{O}(N^{\mathsf{PW}(\mathcal{H})} + Z)$ [11]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
#CQ	$\sum_{(x_1,\dots,x_f)} \max_{x_{f+1}} \cdots \max_{x_n} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\mathsf{DM}(\mathcal{H})} + Z)$ [18]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
Joins	$\bigcup_{\mathbf{x}} \bigcap_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}\left(N^{\mathsf{fhtw}(\mathcal{H})} + Z\right)$ [21]	$\tilde{O}\left(N^{\mathsf{fhtw}(\mathcal{H})} + Z\right)$
Marginal Distribution	$\sum_{(x_{f+1},\dots,x_n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\mathsf{htw}(\varphi)} + Z)$ [23]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
MAP query	$\max_{(x_{f+1},\dots,x_n)} \prod_{S\in\mathcal{E}} \psi_S(\mathbf{x}_S)$	$\tilde{O}(N^{\mathsf{htw}(\varphi)} + Z)$ [23]	$\tilde{O}(N^{\mathsf{faqw}(\varphi)} + Z)$
Matrix Chain Mult.	$\sum_{x_2,\dots,x_n} \prod_{i=1}^{n} \psi_{i,i+1}(x_i, x_{i+1})$	DP bound [15]	DP bound
DFT	$\sum_{(y_0,\dots,y_{m-1})\in\mathbb{Z}_p^m} b_y \cdot \prod_{0\leq j+k<m} e^{i2\pi \frac{x_j \cdot y_k}{p^{m-j-k}}}$	$O(N \log_p N)$ [14]	$O(N \log_p N)$

Table 1: Runtimes of algorithms assuming optimal variable ordering is given. Problems shaded red are in CSPs and logic ($\mathbf{D} = \{0,1\}$ for CSP and $\mathbf{D} = \mathbb{N}$ for #CSP), problems shaded green fall under PGMs ($\mathbf{D} = \mathbb{R}_+$), and problems shaded blue fall under matrix operations ($\mathbf{D} = \mathbb{C}$). N denotes the size of the largest factor (assuming they are represented with the listing format). $\mathsf{htw}(\varphi)$ is the notion of integral cover width for PGM. $\mathsf{PW}(\mathcal{H})$ is the *optimal width of a prefix graph* of $\mathcal{H}$ and $\mathsf{DM}(\mathcal{H}) = \text{poly}(F\text{-}\mathsf{ss}(\mathcal{H}), \mathsf{fhtw}(\mathcal{H}))$, where F-$\mathsf{ss}(\mathcal{H})$ is the $[f]$-quantified star size. Z is the output size in listing representation. Our width $\mathsf{faqw}(\varphi)$ is never worse than any of the three and there are classes of queries where ours is unboundedly better than all three. In DFT, $N = p^m$ is the length of the input vector. $\tilde{O}$ hides a factor of $\text{poly}(|\mathcal{H}|) \cdot \log N$.

makes use of generalizations of AGM to queries with functional dependencies and immaterialized predicates (such as $a+b=c$). These new bounds are based on a linear program whose variables are marginal entropies [4, 5].

The second problem is to select a good variable ordering to run InsideOut on. In principle, one does not have to use the AGM-bound or the bounds from [4, 5] to estimate the cost of an FAQ subquery. If one were to implement InsideOut inside any RDBMS, one could poll that RDBMS's optimizer to figure out the cost of a given variable ordering. However, there are $n!$ variable orderings, and optimizer's cost estimation is time-consuming. Furthermore, some subqueries have inputs which are intermediate results. Hence, it is much faster to compute a variable ordering minimizing the faqw of the query, defined on the bounds in [4, 5]. As the problem is **NP**-hard, either an approximation algorithm [3] or a greedy heuristic suffices in our experience.

InsideOut is bottom-up dynamic programming. We can also solve FAQ queries with top-down (memoized) dynamic programming. In hindsight, this was the approach that Bakibayev et al. [9] and Olteanu and Závodný [32] took to solve FAQ over a single semiring. We can also limit the amount of memoization in a top-down strategy to attain performance gain in some cases [22].

Practical Impact. It is trivial to construct classes of queries on real datasets for which InsideOut-style of algorithms gives arbitrarily large speedups over traditional RDBMSs. In fact, even when dynamic programming does not take effect, the speedup of backtracking search (and thus worst-case optimal algorithms) over traditional query plans is already huge [30]. The impact of the FAQ-framework and the InsideOut algorithm, however, go much beyond these toy queries (even when run on real datasets).

InsideOut is a component of LogicBlox's effort to extend LogiQL to be a probabilistic programming language [10], as part of DARPA's PPAML and MUSE programs. The component the algorithm handles is inference in discrete graphical models.

Learning from the beautiful work of Olteanu and Schleich [31, 34], we realized [27] that InsideOut can be used to train a large class of machine learning models *inside* the database. Our implementation showed orders of magnitude speedup over the traditional data modeler route of exporting the data and running it through `R` or `Python`. These models are trained with different variations of gradient descents, whose (pre-)computation steps are FAQ queries. What is much more interesting than the vanilla FAQ framework we presented above is that, in these applications, we want to compute *many* (in the 100K-range) FAQ queries all at once, making dynamic programming much more crucial to the performance. Another related approach was considered in [25].

5. CONCLUDING REMARKS

The FAQ framework showed that many common computational tasks over a very wide range of domains such as CSPs, machine learning, relational database, logic, and matrix computations can be performed *inside* a database using the same abstraction. The main idea is to blur the line between data and computation, as we use the database to store, compute, and process functions. The glue of the framework is a simple dynamic programming algorithm called InsideOut, which can be cast as a query-rewriting method and thus it is readily implementable within any RDBMS. These ideas are implemented, tested, and validated within the LogicBlox database system. Our theory predicts practice very well, which is a beautiful thing to see.

6. REFERENCES

[1] M. Abo Khamis, H. Q. Ngo, C. Ré, and A. Rudra. Joins via geometric resolutions: Worst-case and beyond. In *PODS*, pages 213–228. ACM, 2015.

[2] M. Abo Khamis, H. Q. Ngo, and A. Rudra. FAQ: questions asked frequently. *CoRR*, abs/1504.04044, 2015.

[3] M. Abo Khamis, H. Q. Ngo, and A. Rudra. FAQ: questions asked frequently. In *PODS*, 2016.

[4] M. Abo Khamis, H. Q. Ngo, and D. Suciu. Computing join queries with functional dependencies. In *PODS*, pages 327–342, 2016.

[5] M. Abo Khamis, H. Q. Ngo, and D. Suciu. What do shannon-type inequalities, submodular width, and disjunctive datalog have to do with one another? In *PODS*, 2017.

[6] S. M. Aji and R. J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[7] M. Aref, B. ten Cate, T. J. Green, B. Kimelfeld, D. Olteanu, E. Pasalic, T. L. Veldhuizen, and G. Washburn. Design and implementation of the LogicBlox system. In *SIGMOD*, 2015.

[8] A. Atserias, M. Grohe, and D. Marx. Size bounds and query plans for relational joins. In *FOCS*, pages 739–748. IEEE Computer Society, 2008.

[9] N. Bakibayev, T. Kociský, D. Olteanu, and J. Zavodny. Aggregation and ordering in factorised databases. *PVLDB*, 6(14):1990–2001, 2013.

[10] V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena. Declarative probabilistic programming with datalog. In *ICDT*, pages 7:1–7:19, 2016.

[11] H. Chen and V. Dalmau. Decomposing quantified conjunctive (or disjunctive) formulas. In *LICS*, 2012.

[12] S. Chu, M. Balazinska, and D. Suciu. From theory to practice: Efficient join query evaluation in a parallel database system. In *SIGMOD*, pages 63–78, 2015.

[13] S. Cohen. User-defined aggregate functions: Bridging theory and practice. In *SIGMOD '06*, pages 49–60.

[14] J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation*, 19:297–301, 1965.

[15] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT Press, 2nd edition, 2001.

[16] M. Davis, G. Logemann, and D. W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[17] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artif. Intell.*, 113(1-2):41–85, 1999.

[18] A. Durand and S. Mengel. Structural tractability of counting of solutions to conjunctive queries. In *ICDT*, pages 81–92, 2013.

[19] W. Fischl, G. Gottlob, and R. Pichler. General and Fractional Hypertree Decompositions: Hard and Easy Cases. *ArXiv e-prints*, Nov. 2016.

[20] S. W. Golomb and L. D. Baumert. Backtrack programming. *J. ACM*, 12(4):516–524, 1965.

[21] M. Grohe and D. Marx. Constraint solving via fractional edge covers. In *SODA*, pages 289–298, 2006.

[22] O. Kalinsky, Y. Etsion, and B. Kimelfeld. Flexible caching in trie joins, 2017. To appear in EDBT.

[23] K. Kask, R. Dechter, J. Larrosa, and A. Dechter. Unifying tree decompositions for reasoning in graphical models. *Artif. Intell.*, 166(1-2), 2005.

[24] D. Koller and N. Friedman. *Probabilistic graphical models.* Adaptive Computation and Machine Learning. MIT Press, 2009. Principles and techniques.

[25] A. Kumar, J. Naughton, and J. M. Patel. Learning generalized linear models over normalized data. In *SIGMOD*, pages 1969–1984. ACM, 2015.

[26] D. Marx. Approximating fractional hypertree width. *ACM Trans. Algorithms*, 6(2):29:1–29:17, Apr. 2010.

[27] H. Q. Ngo, X. Nguyen, D. Olteanu, and M. Schleich. In-database learning with sparse tensors, 2017. Manuscript.

[28] H. Q. Ngo, E. Porat, C. Ré, and A. Rudra. Worst-case optimal join algorithms. In *PODS*, pages 37–48, 2012.

[29] H. Q. Ngo, C. Ré, and A. Rudra. Skew strikes back: New developments in the theory of join algorithms. In *SIGMOD RECORD*, pages 5–16, 2013.

[30] D. T. Nguyen, M. Aref, M. Bravenboer, G. Kollias, H. Q. Ngo, C. Ré, and A. Rudra. Join processing for graph patterns: An old dog with new tricks. In *GRADES*, pages 2:1–2:8, 2015.

[31] D. Olteanu and M. Schleich. F: regression models over factorized views. *PVLDB*, 9(13):1573–1576, 2016.

[32] D. Olteanu and J. Závodný. Size bounds for factorised representations of query results. *ACM Trans. Datab. Syst.*, 40(1), 2015.

[33] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence).* Elsevier Science Inc., 2006.

[34] M. Schleich, D. Olteanu, and R. Ciucanu. Learning linear regression models over factorized joins. In *SIGMOD*, pages 3–18, 2016.

[35] T. L. Veldhuizen. Triejoin: A simple, worst-case optimal join algorithm. In *ICDT*, pages 96–106, 2014.

[36] N. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Canadian AI*, pages 171–178, 1994.

[37] N. L. Zhang and D. Poole. Exploiting causal independence in Bayesian network inference. *J. Artificial Intelligence Res.*, 5:301–328, 1996.